2021.07.24. 교육

나정휘

https://justiceHui.github.io/

목차

귀류법

귀류법

- 명제의 결론이 부정이라고 가정했을 때 모순이 발생함을 보여 원래 명제가 참임을 증명

- 모든 n에 대해 P(n)이 참임을 증명
  - P(n)이 거짓이 되는 n이 있다고 가정
  - P(n)이 거짓이 되는 n이 있으면 모순이 일어나는 것을 보임
  - P(n)이 거짓이 되는 n이 없으므로 모든 n에 대해 P(n)은 참

귀류법의 예시

- √2가 유리수가 아님을 증명
  - √2가 유리수라고 가정하자.
  - 서로소인 자연수 p,q에 대해 √2 = p/q라고 표현할 수 있다.
  - 양변을 제곱하면 2 = p^2 / q^2, 2q^2 = p^2
  - p^2이 짝수이므로 p는 2의 배수
  - 2q^2이 4의 배수이므로 q는 2의 배수
  - p q 모두 2의 배수이므로 서로소가 아님 ⇒ 모순 발생

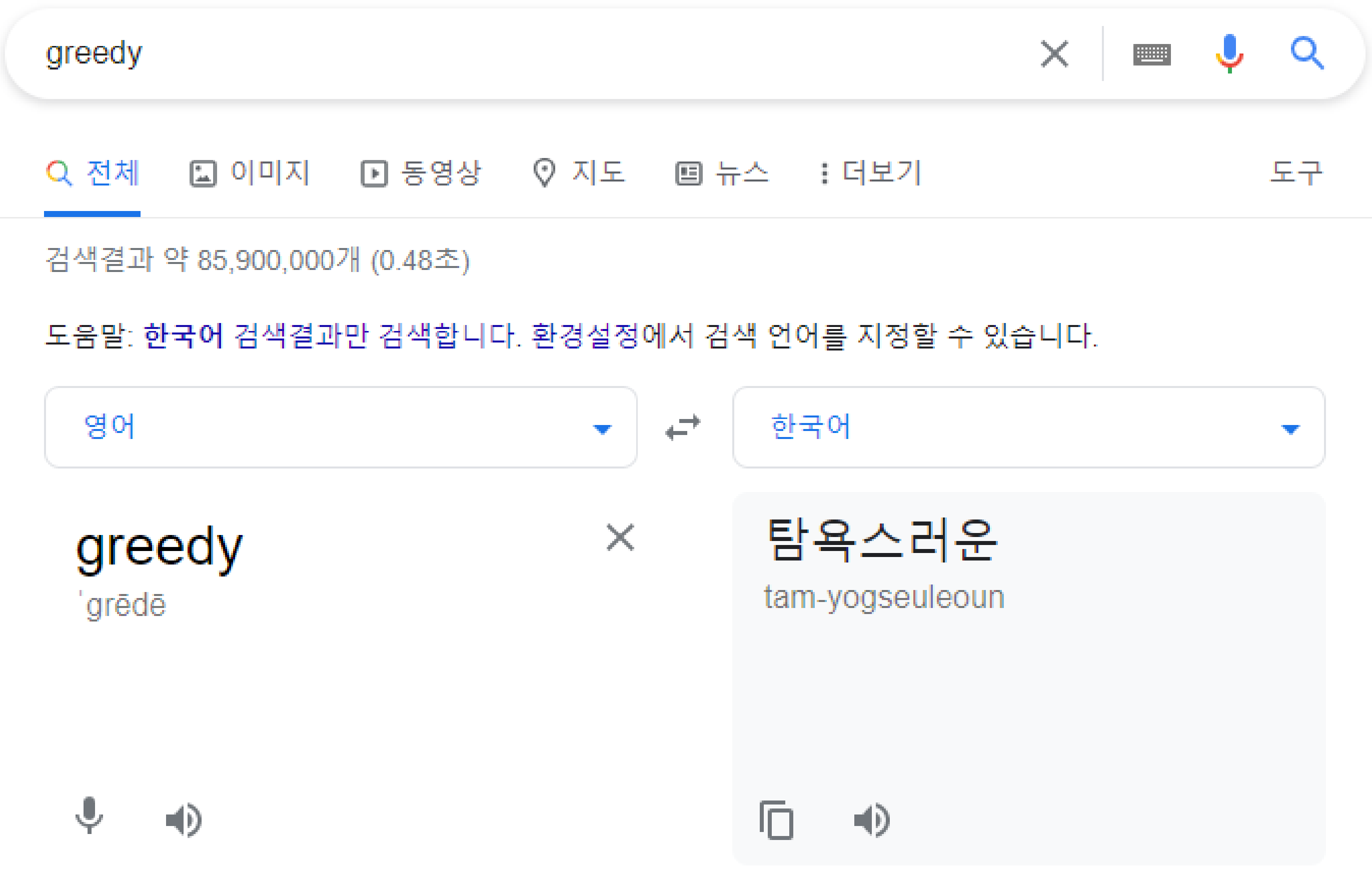
귀류법의 예시

- 소수는 무한히 많음을 증명
  - 소수가 유한하다(k개)고 가정하고, 그 소수를 p1, p2, … , pk라고 하자.
  - n = p1p2…pk + 1을 생각해보자.
  - p1은 n의 약수가 아니다.
  - p2는 n의 약수가 아니다.
  - …
  - pk는 n의 약수가 아니다.
  - n은 모든 소수로 나눠지지 않기 때문에 소수가 되어야 하는데
  - 소수는 k개 밖에 없어야 하므로 모순

질문?

그리디 기법

그리디


greedy
전체
이미지
동영상
지도
뉴스
더보기
도구
검색결과 약 85,900,000개 (0.48초)
도움말: 한국어 검색결과만 검색합니다. 환경설정에서 검색 언어를 지정할 수 있습니다.
영어
한국어
greedy
ˈgrēdē
탐욕스러운
tam-yogseuleoun

그리디

- 현재 상태에서 가장 좋을 것 같은 선택을 함
  - 이 선택이 전체 범위에서도 가장 좋은 선택이라는 것은 보장 못함
  - 만약 이러한 전략이 전체 범위에서도 최적이라면 그리디 알고리즘으로 문제를 해결할 수 있음

- 약간 이런 느낌
  - 동적 계획법: D[i]를 계산할 때 $1 \le j < i$인 모든 j를 확인함
  - 그리디 기법: D[i]를 계산할 때 D[i-1]만 확인함

그리디

- 많이 보이는 형태
  - N개의 원소가 주어짐
  - 특정 조건을 만족하도록 원소 몇 개를 선택해서
  - 원소의 가중치의 합을 최대/최소화하는 문제

```
A[N] = 원소들
Res[] = 정답 집합

sort(A, A+N); // A를 특정 기준으로 정렬
for(int i=0; i<N; i++){
    if(Feasible(Res + A[i])){ // 정답에 A[i]를 추가해도 조건을 만족하면
        Res += A[i];
    }
}
```

그리디 기법의 예시 1

BOJ 11047 동전 0

- N가지의 동전을 사용해서 K원을 만듬
- 사용하는 동전의 개수를 최소화
- 동전은 서로 약수/배수 관계, 1원 짜리 동전은 항상 존재

- 가격이 큰 것부터 사용하는 그리디

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, K, A[11], R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=N; i>=1; i--){
        R += K / A[i];
        K %= A[i];
    }
    cout << R;
}
```

BOJ 11047 동전 0 – 증명

- 동전을 가격에 대한 내림차순으로 정렬하자 : C[i] = i번째로 비싼 동전
- 그리디에서 각 동전을 사용한 개수를 저장한 리스트 A
- 실제 최적해에서 각 동전을 사용한 개수를 저장한 리스트 B

- (귀류법) A보다 더 적은 동전을 사용하는 **최적해** B가 존재한다.
  - A와 B가 처음으로 달라지는 지점 i가 존재할 것이다.
  - 비싼 동전부터 최대한 많이 가져가는 그리디 알고리즘에 의해 A[i] > B[i]
  - C[i] * (A[i] – B[i])원 만큼의 차이를 채우기 위해 C[i]보다 싼 동전을 더 사용할텐데
  - C[i] * (A[i] – B[i])는 C[i]의 배수이고
  - C[i]보다 싼 동전으로 C[i] * (A[i] – B[i])원을 만드는데 A[i]–B[i]보다 더 많은 동전이 필요하므로
  - B는 A보다 동전을 더 적게 사용할 수 없다.

질문?

0/1 Knapsack Problem

- N개의 물건과 최대 X kg까지 넣을 수 있는 배낭이 있다.
- i번째 물건의 무게는 Wi kg이고, 가격은 Pi원이다.
- 무게의 합이 X kg 이하가 되도록 배낭에 물건을 넣을 때
- 가능한 가격의 최댓값을 구하는 문제

- 이건 동적 계획법으로 풀림
- 물건을 분할할 수 있다면?

Fractional Knapsack Problem

- 물건을 분할할 수 있음
  - 무게가 Wi, 가격이 Pi인 물건을 무게가 x, Wi-x인 물건을 분할하면
  - (x, Pi/Wi*x), (Wi-x, Pi/Wi*(Wi-x))로 나뉨

- 무게 대비 가격(효율)이 높은 물건부터 가져가는 그리디

Fractional Knapsack Problem

```
#include <bits/stdc++.h>
using namespace std;

struct Element{
    int W, P;
    Element() = default;
    Element(int W, int P) : W(W), P(P) {}
};

int N, X;
vector<Element> elements;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> X;
    for(int i=0; i<N; i++){
        int W, P; cin >> W >> P;
        if(P > 0) elements.emplace_back(W, P);
    }

    sort(elements.begin(), elements.end(), [](const Element &e1, const Element &e2){
        return e1.P * e2.W > e2.P * e1.W; // e1.P / e1.W > e2.P / e2.W
    });

    int weight = 0;
    double price = 0;
    for(auto [w,p] : elements){
        if(weight < X){
            int take = min(X - weight, w);
            weight += take;
            price += 1.0 * p / w * take;
        }
    }
    cout << price;
}
```

0/1 Knapsack Problem – 반례

- 0/1 Knapsack Problem에서는 이 전략이 안 통할까?
  - N = 4, X = 3, (Wi,Pi) = { (1,3), (3,7), (1,2), (1,1) }
  - 최적해 : { (3,7) }
  - 그리디 알고리즘이 구한 해 : { (1,3), (1,2), (1,1) }
  - 안 된다.

Fractional Knapsack Problem – 증명

- 가격이 0 이하인 경우 신경 안 써도 됨
- 무게의 합이 X 이하인 경우 모두 가져가는 것이 최적
  - 그리디 알고리즘은 이 경우를 잘 처리함
- 그렇지 않은 경우 무게를 X로 꽉 채우는 것이 이득
  - 그리디 알고리즘은 이 경우 무게를 X로 꽉 채움

- 최적해의 무게 합이 X라고 가정하자.
- 증명의 편의를 위해 모든 물건의 효율이 다르다고 가정하자.

Fractional Knapsack Problem – 증명

- 물건을 효율에 대한 내림차순으로 정렬하자.
- 그리디에서 각 물건을 가져간 무게를 저장한 리스트 A
- 실제 최적해에서 각 물건을 가져간 무게를 저장한 리스트 B

- (귀류법) A보다 가격이 더 비싼 **최적해** B가 존재한다.
  - A와 B가 처음으로 달라지는 지점 i가 존재할 것이다.
  - 효율이 높은 것부터 최대한 많이 가져가는 그리디 알고리즘에 의해 A[i] > B[i]이다.
  - B는 최적해이기 때문에 A[j] < B[j]인 j(> i)가 존재한다.
  - 새로운 해 B′를 구성한다. B′[i] = B[i] + eps, B′[j] = B[j] – eps이고, 다른 모든 B′[k] = B[k]이다.
  - B와 B′의 무게는 동일하지만, 가격은 B′가 더 비싸다.
  - B가 최적해라는 가정에 모순이 생겼으므로 A보다 더 비싼 최적해는 존재하지 않는다.

질문?

BOJ 11399 ATM

- i번째 프로세스는 CPU를 Pi만큼 점유함
- 대기 시간의 합을 최소로 하는 스케줄링을 찾는 문제

- 1학년 컴퓨터 시스템 일반 과목
  - “SJF 스케줄링은 평균 대기 시간을 최소화한다.”
  - 평균 대기 시간 최소 ⇒ 대기 시간 합 최소

- 소요 시간이 짧은 프로세스부터 수행하면 됨

BOJ 11399 ATM – 증명

- 두 원소 P_{k}, P_{k+1} 중 먼저 와야 하는 원소를 결정하자.
- 인접한 원소의 순서만 잘 결정할 수 있다면, 버블 정렬 느낌으로 전체 원소 정렬 가능

P_{1..k-1}	P_{k}	P_{k+1}	P_{k+1..n}

P_{1..k-1}	P_{k+1}	P_{k}	P_{k+1..n}

- P_{1..k-1}과 P_{k+1..n}의 대기 시간은 동일함 / P_{k}와 P_{k+1}의 대기 시간만 보면 됨
- 위 : sum(P_{1..k-1}) + sum(P_{1..k-1}) + P_{k}
- 아래 : sum(P_{1..k-1}) + sum(P_{1..k-1}) + P_{k+1}
- 동류항 날리면 위에는 P_{k}, 아래에는 P_{k+1}만 남음
- 그러므로 P_{k}가 작은 것이 먼저 오도록 정렬하는 것이 이득

질문?

BOJ 1931 회의실 배정

- 한 개의 회의실이 있음
- 각 회의는 시작 시간과 종료 시간이 있음
- 몇 개의 회의를 선택해서 서로 겹치지 않도록 회의를 진행할 때
- 진행 가능한 최대 회의 개수

- 끝나는 시간이 빠른 것부터 정렬하고
- 가능한 회의를 진행하는 그리디가 성립함

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;

int N;
PII A[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    sort(A+1, A+N+1, [](PII p1, PII p2){
        if(p1.y != p2.y) return p1.y < p2.y;
        return p1.x < p2.x;
    });

    int mx = 0, cnt = 0;
    for(int i=1; i<=N; i++){
        if(mx <= A[i].x) cnt++, mx = A[i].y;
    }
    cout << cnt;
}
```

BOJ 1931 회의실 배정 – 증명

- Claim) 종료 시간이 가장 빠른 회의를 포함하는 최적해가 존재한다.
  - Proof) 종료 시간이 가장 빠른 회의 mn를 포함하지 않는 최적해 O가 존재한다고 하자.
  - 당연히 O에는 겹치는 회의가 존재하지 않는다.
  - O에서 종료 시간이 가장 빠른 회의 x를 제거하고 mn을 추가한 O-x+mn을 생각해보자.
  - O-x+mn의 크기는 O와 동일하고, O-x+mn에는 겹치는 회의가 존재하지 않는다.
  - 그러므로 종료 시간이 가장 빠른 회의를 포함하는 최적해가 항상 존재한다.

질문?

정리

- 지금까지 본 증명 방법
  - A보다 좋은 최적해 B가 있다고 가정한 뒤, B가 최적해가 아님을 보임
    - Example) 동전 0, Fractional Knapsack Problem
  - 원소의 우선순위를 정한 뒤, 우선순위가 높은 것부터 선택하는 것이 최적임을 보임
    - Example) ATM(SJF Scheduling)
  - 어떤 원소를 포함하는 최적해가 항상 존재함을 보임
    - Example) 회의실 배정

Exchange Argument

Exchange Argument

- BOJ 11399 ATM의 증명을 일반화한 것

- 인접한 두 원소의 순서를 결정할 수 있으면
- 버블 정렬 느낌으로 전체 원소의 순서를 결정할 수 있음

- 버블 정렬과 다른 일반적인 정렬의 결과물은 동일하므로
- 비교 함수를 잘 작성한 뒤 std::sort를 사용하면 됨

BOJ 14908 구두 수선공

- i번째 작업을 수행하는데 T_i일 걸림
- i번째 작업의 완료가 하루 지연될 때마다 S_i원 벌금 내야 함
- 벌금을 최소로 하는 작업 순서를 정하는 문제

BOJ 14908 구두 수선공

- sum(T_{1..k-1}) * S_{k} + (sum(T_{1..k-1}) + T_{k}) * S_{k+1}
- sum(T_{1-k-1}) * S_{k+1} + (sum(T_{k..k-1}) + T_{k+1}) * S_{k}

T_{1..k-1}	T_{k}	T_{k+1}	T_{k+1..n}
T_{1..k-1}	T_{k+1}	T_{k}	T_{k+1..n}

- T_{k} * S_{k+1} ≤ T_{k+1} * S_{k}가 되도록 정렬해야 함
- T{k} / S_{k} ≤ T_{k+1} / S_{k+1}이 되도록 정렬해야 함
- T_{i} / S_{i} 오름차순 정렬

BOJ 2180 소방서의 고민

- 함수 f_i(x) = a_i * x + b_i가 여러 개 주어짐 (a_i, b_i > 0)
- x = 0에서 시작해서 x = x + f_i(x)를 한 번씩 적용할 때
- 최종 결과의 최솟값을 구하는 문제

BOJ 2180 소방서의 고민

- (a_{k+1} + 1) * ((a_{k} + 1) * x + b_{k}) + b_{k+1}
- (a_{k} + 1) * ((a_{k+1} + 1) * x + b_{k+1}) + b_{k}

x	a_{k} * x + b_{k}	a_{k+1} * x + b{k+1}	f

x	a_{k+1} * x + b{k+1}	a_{k} * x + b_{k}	f

- a_{k+1} * x + a_{k} * a_{k+1} * x + a_{k+1} * b_{k} + x + a_{k} * x + b_{k} + b_{k+1}
- a_{k} * x + a_{k} * a_{k+1} * x + a_{k} * b_{k+1} + x + a_{k+1} * x + b_{k+1} + b_{k}
- a_{k+1} * b_{k} ≤ a_{k} * b_{k+1}이 되도록 정렬
- b_{k} / a_{k} ≤ b_{k+1} / a_{k+1}이 되도록 정렬
- b_{i} / a_{i} 오름차순 정렬

질문?

Minimum Spanning Tree

Minimum Spanning Tree

- 무방향 가중치 연결 그래프
- N개의 정점, M개의 간선
- 모든 정점들이 연결되도록 간선 N-1개를 선택할 때
- 간선의 가중치 합을 최소화하는 문제

Minimum Spanning Tree

- Kruskal's Algorithm
  - 간선을 가중치 오름차순으로 정렬
  - 간선을 차례대로 보면서
  - 만약 간선을 추가했을 때 사이클이 안 생기면 해당 간선 추가

- 항상 최소 신장 트리를 찾을 수 있음

Minimum Spanning Tree – 증명

- **Lemma 1.** 같은 정점 집합에서 정의된 두 포레스트 F1 = (V, E1), F2 = (V, E2)에서 |E1| < |E2|이면 F3 = (V, E1+e)가 포레스트가 되도록 하는 e가 E2-E1에 존재한다.
  - 일단 증명 없이 받아들이자.
- Kruskal’s Algorithm에서 사용한 간선들의 리스트 A (가중치 내림차순)
- 실제 최적해에서 사용한 간선들의 리스트 B (가중치 내림차순)

- (귀류법) A보다 가중치가 작은 신장 트리 B가 존재한다.
  - 처음으로 W(A[i]) > W(B[i])가 되는 i를 생각해보자.
  - A, B의 부분 배열 A’ = { A[1], A[2], … , A[i-1] }, B’ = { B[1], B[2], … , B[i] }에서
  - A’, B’는 포레스트이므로 A’ + e가 포레스트가 되도록 하는 e가 B’-A’에 존재한다.
  - W(A[i]) > W(B[i]) ≥ W(e)이므로 Kruskal’s Algorithm은 A[i] 대신 e를 선택한다.
  - 그러므로 Kruskal’s Algorithm은 A를 만들 수 없다.

질문?

BOJ 20761 초직사각형

- 각 변의 길이가 A, B, C, D인 4차원 초직사각형이 있음
- 특정 변의 길이를 일정량 증가시킬 수 있는 카드 N개 주어짐
- 그 중 K개를 사용해서 초직사각형의 부피를 최대화

BOJ 20761 초직사각형

- 같은 변을 늘리는 카드는 증가량이 큰 것부터 사용하는 것이 이득인 것은 자명함
- 각 카드의 증가율을 계산하자
  - 같은 변을 늘리는 카드를 다 모은 다음 내림차순 정렬한 뒤, Prefix Sum을 계산하면 쉬움

- 증가율이 큰 카드부터 사용하는 그리디가 성립한다.

BOJ 20761 초직사각형 – 증명

- 그리디로 선택한 카드들의 증가율 리스트 A (내림차순)
- 실제 최적해의 증가율 리스트 B (내림차순)

- (귀류법) A보다 더 좋은 최적해 B가 존재한다고 가정하자.
  - B의 결과가 더 크기 위해서는 A[i] < B[i]인 i가 존재해야 한다.
  - 그리디 알고리즘은 증가율이 큰 것부터 선택하기 때문에 항상 A[i] ≥ B[i]이다.

- A, B, C, D와 카드의 증가율에 모두 로그를 씌우면 곱셈 대신 덧셈으로 계산할 수 있다.
- 이렇게 하면 증가율이 큰 것부터 가져가야 한다는 것이 조금 더 명확하게 보인다.

더 공부할 거리

- Matroid
  - MST : Graphic Matroid에서의 Minimum Weight Independent Set
  - 초직사각형 : 로그 씌우면 Uniform Matroid에서의 Maximum Weight Independent Set

6/7월 교육 끝

- 수고하셨습니다.